Hardware-Aware Quantization for Multiplierless Neural Network Controllers

Tobias Habermann, Jonas Kühle, Martin Kumm, Anastasia Volkova

Hardware-Aware Quantization for Multiplierless Neural Network Controllers

Tobias Habermann[†], Jonas Kühle[†], Martin Kumm[†], Anastasia Volkova[*]
[†]Fulda University of Applied Science, Fulda, Germany
[*]Nantes Université, CNRS, LS2N, 44000 Nantes, France
{tobias.habermann, jonas.kuehle, martin.kumm}@cs.hs-fulda.de, anastasia.volkova@univ-nantes.fr

***Abstract*—Deep neural networks (DNNs) have been successfully applied to the approximation of non-linear control systems. These DNNs, deployed in safety-critical embedded systems, are relatively small but require a high throughput. Our goal is to perform a coefficient quantization to reduce the arithmetic complexity while maintaining an inference with high numerical accuracy. The key idea is to target multiplierless parallel architectures, where constant multiplications are replaced by bit-shifts and additions. We propose an adder-aware training that finds the quantized fixed-point coefficients minimizing the number of adders and thus improving the area, latency and power. With this approach, we demonstrate that an automatic cruise control floating-point DNN can be retrained to have only power-of-two coefficients, while maintaining a similar mean squared error (MSE) and formally satisfying a safety check. We provide a push-button training and implementation framework, automatically generating the VHDL code.**

***Index Terms*—neural network controllers, machine learning, hardware-aware quantization, quantization-aware training**

I. Introduction

Deep Neural Networks (DNNs) are now applied in practically every domain of human activity. Their size varies depending on applications, from small models having several hundreds or thousands of parameters for regression tasks and control, up to huge compositions of models with billions of parameters for natural language processing. While energy-efficiency and acceleration is of interest for any DNN, in this paper we will focus on inference for small feed-forward neural networks with applications in control. Indeed, Neural Network Controllers (NNCs) are typically implemented with just a few hidden layers and their goal is to serve as a cheaper, faster and more robust substitute to classic numerical control algorithms [1]–[3]. A typical approach is to use High Level Synthesis (HLS) to generate Field-Programmable Gate Array (FPGA) implementations, which do not necessarily take advantage of the structured nature of DNNs or satisfy the high-throughput and low-power needs of embedded controllers in safety-critical applications. With this work, our goal is to remedy for that and propose a new push-button tool for an automatic generation of FPGA-based inference circuits for small but efficient and fast DNNs.

The most costly operations in the realization of feed-forward DNN are the multiplications. In this work we leverage the fact that DNN weights are constants, known a priori, hence generic multiplications can be replaced by additions, subtractions and bit-shifts. These multiplierless implementations yield smaller, faster and less power-hungry implementations. However, simply deducing a multiplierless implementation upon straightforwardly-quantized weights/biases is not satisfactory: constants that are close to each other, in absolute value, can have a significantly different hardware implementation cost. Take for example, integers 31 and 32: multiplying by 31 costs one addition $(31x = 2^5x - x)$ while multiplication by 32 is a simple bit-shift $(32x = 2^5x)$, which can be hardwired at negligible cost. As a consequence, in [4] Gustafsson et al. proposed an *Addition Aware Quantization* approach, in which floating-point numbers are quantized s.t. their multiplierless implementation cost is small.

It has now become a common approach to re-train the floating-point (FP) DNNs after a quantization to low word length fixed-point (FxP) representation, in order to improve the network's accuracy [5]. During a typical re-training, the weights and biases are kept in the chosen, and often uniform, low word length and can take any value, which is not necessarily hardware-friendly. In the previous work [6], a quantization-aware training was proposed, in which FP neural networks are retrained for FxP coefficients that are *selected from a restricted set of numbers* that are well-suited for reconfigurable multipliers.

Building up on this idea, we propose a novel hardware-oriented training scheme, in which FxP weights and biases are selected from the sets of hardware-friendly numbers well-suited for multiplierless implementations with very few adders per quantity. This adder-aware strategy yields significantly faster architectures, as the computations are done fully in parallel.

II. Neural Network Controllers

The application of DNNs to control systems is an emerging domain showing excellent results. Neural networks have demonstrated two main advantages over classic methods: speed and robustness. As shown in case of autonomous driving [2], [7], NNCs can learn how to optimize all decision steps simultaneously, instead of multi-step linking of a classic object detection and control algorithms. Most importantly, NNCs often allow to compress the numerical schemes classically used for control. For instance, in case of aircraft collision avoidance systems, [1] showed that teaching a small DNN to approximate the result of a Markov decision processing reduces the memory footprint for the implemented system by

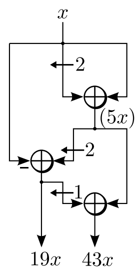

Fig. 1: Shift-and-add based MCM circuit for multiplication with constants 19 and 43

a factor of 1000, compared to classic approaches, and even surpasses the original in robustness.

One crucial property of the NNCs is the inference speed, as these control tasks would be implemented in real-time embedded systems. For instance, [3] focuses on providing small DNN models for fast object-detection, reducing the typical size of DNNs to several, up to 10, dense/convolutional layers with ReLU or $\tanh$ activation functions.

Although utilizing DNNs in safety-critical applications can demonstrate considerable performance benefits, assuring the safety and robustness of these systems is indispensable. Moreover, it has been demonstrated that their behavior can be unpredictable due to adversarial perturbations in their inputs. Hence, a formal set-based verification tool, such as Neural Network Verification (NNV) [8], should be employed to guarantee the safety of the *implemented* neural controller.

III. Multiple Constant Multiplication

One efficient way for realizing MCM circuits is to transform the required multiplications into several additions, subtractions and bit-shifts (shift-and-add, for short). Fig. 1 shows an example of an MCM operation with the two constants 19 and 43. Arrows denote bit-shifts, and each adder can be assigned to a constant factor. For example, the first adder computes $5x$ by adding x to x bit-shifted by two, leading to $x + 2^2x = 5x$. Similar, $19x$ is computed as $2^2(5x) - x = 19x$. Instead of two multipliers, this MCM implementation only requires the hardware equivalent of three adders.

Determining the circuit with the least amount of adders for a given set of constants is known as the MCM problem. Several heuristics [9]–[12] and optimal approaches [13]–[15] have been presented to solve it.

To reduce the hardware complexity of our NN controller, we can simply apply these algorithms to the quantized weights of our NN controller. However, the complexity (in terms of adder count) strongly depends on the coefficient value. For example, the multiplication with the factor 43 is known to require three adders. The multiplication with 42 (or 44) would require only two adders. Quantizing a real number to those cheaper numbers was previously proposed as addition-aware quantization [4].

We go one step further and make use of the robustness of the neural network: we quantize the weights of the NN controller to a set of cost-n numbers, for which the adder count does not exceed n adders. However, such quantization induces potentially large errors, perturbing the algorithm's output. We remedy for that by re-training the system with a new process called adder-aware training (AAT).

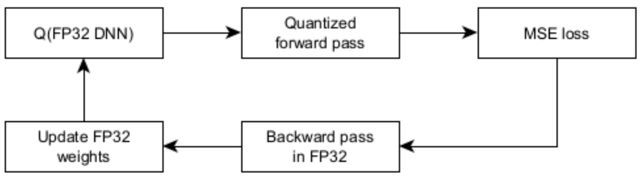

Fig. 2: The proposed training flow. Function Q is a quantization strategy algorithm.

IV. Adder-Aware Training

The quantization of DNNs is an active research area, where various different techniques were explored in recent years [5]. The subject can be split into two main streams. First, the post training quantization (PTQ) where the trained FP DNN is quantized to FxP formats. And second, the quantization-aware training (QAT) where the FP DNN is quantized and then is re-trained. In most cases, PTQ is faster and its advantage is that it does not require a training dataset, which is often unavailable. However, the results are not as good as with QAT methods.

In this work, our goal is to leverage the power of MCM and implement hardware inference using the optimized multiplierless circuits. Based on the general observation of the exceptional robustness of DNNs w.r.t. quantization, we postulate that weights and biases can be quantized to very simple low-precision numbers, that can be implemented with only a few adders. By fixing the hardware cost of each weight to be 0, 1 or 2 adders, we restrict the set of possible values that can be taken and, hence, the QAT should be adapted accordingly. We integrated our work in the open-source Modular Quantization Aware Training (MQUAT) framework, which is based on the ideas proposed in [6]. It provides a framework for QAT on limited sets of possible numbers for weights and biases.

The general proposed training flow is roughly illustrated in Fig. 2. Our framework takes the following information as inputs: the FP32 DNN model, the training and validation sets, the hyperparameters, e.g., learning rate and number of epochs, and the description of the hardware-cost restriction for all weights, layer- or channel-wise. The pre-trained FP32 DNN is re-trained by first finding a proper integer and fractional bit distribution for each unit to quantize. Then, the training process is started, where all weight updates are applied to the full precision weights but calculated based on the quantized DNN. After each backward pass, where weights and biases are updated in FP32, the DNN's coefficients are quantized. When the quantization function Q is a simple rounding, this process is a classic QAT. In the proposed AAT, we transform this function into a more complex algorithm, in which FP32 weights are rounded to the closest numbers that are realizable with the restricted adder cost.

First, we set up two types of quantization: a channel-wise quantization, in which each channel in a layer has its own FxP format, and a layer-wise quantization, in which all channels in the layer share the same format and, in consequence, the same shift-and-add graph. Channel-wise quantization allows tailoring the quantized channels better to the original ones and utilizes the whole potential of a much faster fully-parallel implementation of the DNN. Layer-wise quantization is interesting for resource-saving strategy but is typically slower, since it requires a sequential sharing of adders.

Due to the highly discrete nature of the target sets, from, which each coefficient value is selected, the convergence of the adder-aware training is not guaranteed. As the numerical experiments shall demonstrate, when the training reaches the user-given number of re-training epochs, the mean squared error (MSE) loss of the quantized DNN can be too high. In this case, a higher hardware cost, which increases the size of the coefficient set K, should be considered.

The quantization function $Q(v)$ itself becomes a selection procedure, in which for each FP constant v, we search for an element k in the coefficient set K such that $\min_{k\in K}|k-v|$. It might happen, that the coefficient set K has to be shifted to provide a lower quantization error.

There are several coefficient sets $K \in \mathbb{K}$ possible that are constructed according to the target adder-cost and the target word length: The cost-0 set contains coefficients that are all powers of two, i.e., implementable uniquely with bit-shifts. For example, for 5-bit representation $K_0^5 = \{-32, -16, \ldots, 0, \ldots, 16, 32\}$. Formally, the set of all sets for cost-0 l-bit numbers $\mathbb{K}_0^l$ always contains just one coefficient set and is defined as

$$\begin{aligned} K_0^l &= \{\pm(2^n) \,:\, \forall n = 0 \ldots l\} \cup \{0\} \\ \mathbb{K}_0^l &= \{K_0^l\}\ . \end{aligned}$$

The cost-1 sets for l-bit collects all coefficient sets obtained through a sum of two cost-0 l-bit coefficients. Similarly, the cost-2 set of all sets for l-bit representation collects all coefficient sets that can be represented through a shifted sum of two individual sums of two cost-1 l-bit coefficients. We define the set of sets $\mathbb{K}_1^l$ and $\mathbb{K}_2^l$ with the help of the set of base combinations sets $\mathbb{B}_1^l$, $\mathbb{B}_2^l$ and a clipping function $\text{clip}_l(x) = \min(2^l, \max(-(2^l), x))$ in the following way:

$$\begin{aligned} \mathbb{B}_1^l &= \{\{1, k_a + k_b\} : \forall K \in \mathbb{K}_0^l, \forall k_a, k_b \in K\} \\ \mathbb{B}_2^l &= \{\{1, k_a + k_b\} : \forall K_1 \in \mathbb{K}_1^l, \forall K_0 \in \mathbb{K}_0^l, \forall k_a \in K_0, \forall k_b \in K_1\} \\ &\quad \cup \{\{1, k_a + k_b, k_c + k_d\} : \forall K \in \mathbb{K}_0^l, \forall k_a, k_b, k_c, k_d \in K\} \\ \mathbb{K}_1^l &= \{\{\pm\text{clip}_l(b \cdot 2^n) \,:\, \forall n = 0..l,\, \forall b \in B\} : \forall B \in \mathbb{B}_1^l\} \\ \mathbb{K}_2^l &= \{\{\pm\text{clip}_l(b \cdot 2^n) \,:\, \forall n = 0..l,\, \forall b \in B\} \,:\, \forall B \in \mathbb{B}_2^l\} \end{aligned}$$

In the quantization function Q we encode the search for the best fitting set $K \in \mathbb{K}_n^l$, for the target FP value v according to the absolute error between the value v and the candidate set. For a channel-wise quantization, each channel per layer has its own cost-n set. For the layer-wise quantization, a target set K is shared between each layer. The target set is selected based on the sum of absolute differences between the coefficients and the elements of the candidate set.

V. Experimental evaluation

A. Hardware Architecture Generator

We have integrated our VHDL code generator for the proposed multiplierless fully-parallel feed-forward DNNs in the open-source core generator framework FloPoCo[1] [16]. Given a tensor-flow model in Python and the quantized coefficients, it generates the hardware architecture. Each channel is computed by first using a multiplierless approach for partial products, and then by adding all signals in a compressor tree. For partial products of a negative weight, the product with the positive weight is computed in a multiplierless way and the result is subtracted in the compressor tree. All the data paths are automatically truncated and resized to an internal FxP format. We leave the complex task of error-analysis and impact of the truncations upon the accuracy of the neural network out of scope of this paper. Instead, we compute all the signals exactly, but truncate the outputs of the activation layers to a user-given value.

B. DNN-based Adaptive Cruise Control

We evaluate the proposed approach on an NNC that implements an adaptive cruise control (ACC) task. This example comes from the NNV toolbox demonstration [8] for safety verification of NNCs and has been used for the ARCH competition [17]. An ACC is a system used in cars to keep a save distance between itself (the ego car) and the leading car. The ACC is provided with the current velocities and positions of both cars and has to calculate the acceleration of the ego car to keep a pre-defined safe distance. The ideal solution for the control problem is computed via a differential equation, which actually provides the training and validation data for the DNN model that approximates the solution of the equation. We consider the ACC3 model from the ARCH competition, which is a simple feed-forward network with 3 hidden layers, each having 20 neurons with ReLU activation function, and one last additional layer having one linear neuron to match the output format. We evaluate the MSE on the validation data set provided by the differential equation during our adder-aware training. However, a small MSE does not necessarily mean that the controller is actually safe. Hence, in addition, we will also use the safety verification provided by the NNV toolbox to analyze the quantizations of the trained model.

C. Experimental results and discussion

The proposed adder-aware training was integrated into the open-source tool MQUAT, which is written in Python and based on Tensorflow. In all experiments we used the standard Adam optimizer in Tensorflow with a learning rate of 10^{-6}, `amsgrad=True` with 35 training epochs using the standard MSE loss. The training and validation dataset each have 210k

[1]available online at http://flopoco.org

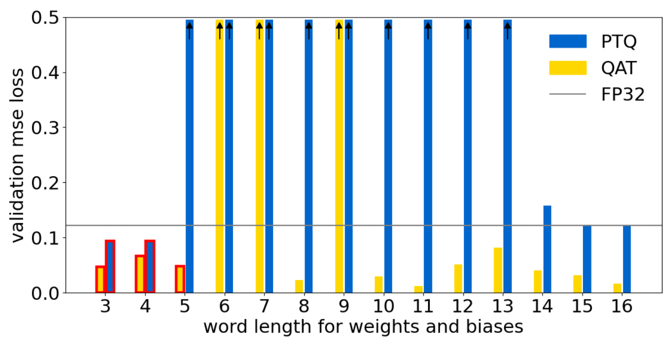

Fig. 3: MSE results for the ACC3 with PTQ and QAT for different FxP word lengths. All results with an MSE higher than 0.5 are marked with an arrow. All unsafe controllers have a red border.

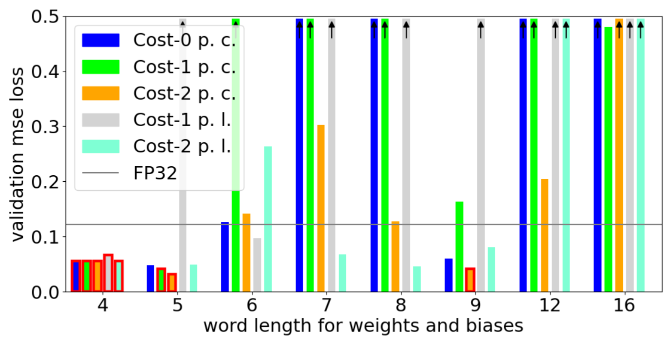

Fig. 4: MSE results for the adder-aware training of the ACC3 controller for different FxP word lengths and target channel- or layer-wise costs. All results with a higher MSE than 0.5 are marked with an arrow. Unsafe controllers are marked red.

examples. The generated hardware implementations are synthesized using Vivado v2018.3 for the xczu19eg-ffvb1517-1 FPGA. The run times are reported for training on a machine with a GeForce RTX 2060 Super GPU with 8GB GDDR6 RAM, i7-4790 CPU and with 16 GB of system memory.

In our first experiment, our goal is to compare the PTQ and the QAT approaches for the chosen ACC3 DNN. Fig. 3 illustrates the MSE of the PTQ (in blue) and the QAT (in yellow) of the reference FP32 network to different FxP word lengths (from 3 to 16). The red borders indicate quantized instances that are proven unsafe by the NNV, and bars marked with an arrow have an MSE >0.5, making the networks unexploitable. It is easy to see that retraining the network after the quantization is extremely important, not only permitting to maintain the MSE for low-precision coefficients but even improving it. The PTQ yields a comparable MSE and a safe controller with 14 bits word length. However, if we apply the quantization-aware training and lower gradually the word length, we can go as low as 8 bits, which we take as a baseline for comparison with our approach.

To obtain a state-of-the-art multiplierless reference, we applied the MCM algorithm RPAG [12] to the baseline 8 bit model. The upper part of Table I illustrates that 165 adders are required in total to implement the baseline controller, which is high but expected, as the weights and biases were not specifically designed for a multiplierless hardware. Simply rounding the coefficients to the closest cost-0, cost-1, etc. numbers (using [4]) yielded unsafe controllers with an MSE larger than 1, confirming our intuition that re-training for adder-aware weights and biases is crucial.

TABLE I: MCM cost for the baseline and proposed approaches

Experiments		**#adders per layer**				**total #adders**
		1	2	3	4	
8 bit baseline		59	46	58	2	**165**
5 bit proposed	cost-0 / channel	0	0	0	0	**0**
	cost-1 / channel	14	10	12	1	**37**
	cost-2 / channel	15	14	14	1	**44**
	cost-1 / layer	1	1	1	1	**4**
	cost-2 / layer	2	2	2	1	**7**

Fig. 4 illustrates the results when applying the proposed adder-aware training for the cost-0, cost-1 and cost-2 coefficient sets. AAT was applied either per channel (p. c.) or per layer (p. l.). The red boxes indicate low-precision instances that, regardless of a very low MSE, are certified unsafe by the NNV tool. Our first observation is that for a word length of just 5 bits our tool finds coefficients of a safe cost-0 controller with an MSE lower than the reference FP32 DNN. This MSE improvement is probably due to a smaller over-fitting in the low-precision DNN. Interestingly, we observed in all our experiments that even if a lower-cost implementation is of a good quality, a higher-cost result is not necessarily as good. For instance, for 5 bit coefficients, only the cost-0 channel-wise and cost-2 layer-wise results are stable, and for word length 6 bits, the MSE for cost-0 is lower than for cost-2 channel-wise. These effects could be explained by the non-deterministic nature of the stochastic gradient descent algorithm used for training, and that falling into local minima is highly probable in ultra low precisions over restricted coefficient sets.

Surprisingly, the larger word sizes (like 12 and 16 bits in Fig. 4) perform worse than the lower precisions (like 9, 6 or 5 bits). The reason for this could be that the chosen pre-trained QAT DNNs for AAT also had a bigger word length leading to a worse fit of the coefficient distribution.

Our conclusion is that a thorough design-space exploration is needed to find the lowest coefficient word length that provides a safe implementation at the lowest adder cost. This is without a doubt possible using the proposed tool, since the running times are reasonable for a one-time analysis. Table II illustrates the running times for layer-wise and channel-wise PTQ and QAT approaches applied on the ACC3 DNN for quantizations of 4, 8 and 16 bits. The classical FP32 to FxP is used as the baseline and shows that the timings are comparable to those of the proposed approach. We can note, that our performance is stable across the word lengths and varying from roughly 4 to 6 minutes for cost-0 coefficients up to 18 minutes for cost-2 coefficients.

In the following, we aim at comparing the resource benefits for different target cost settings, and therefore permit ourselves

TABLE II: Quantization time in seconds for classic FP32 to FxP quantization, for the ACC3 DNN.

	Proposed adder-aware training					Baseline	
	layer-wise		channel-wise			channel-wise	
	$\mathbb{K}_1^l$	$\mathbb{K}_2^l$	$\mathbb{K}_0^l$	$\mathbb{K}_1^l$	$\mathbb{K}_2^l$	PTQ	QAT
4 bit	387	564	356	355	1133	1312	2546
8 bit	387	567	358	370	1116	52	1265
16 bit	391	578	361	373	1139	51	1265

TABLE III: Synthesis results for the baseline and proposed parallel architectures

Design	Experiment	#LUT	#DSP	delay (ns)	MSE
baseline (8 bit)	generic mult	44328	388	12.489ns	0.0272
	generic no DSP	69058	0	11.842ns	0.0272
	shift-and-add [12]	42128	0	7.952ns	0.0272
proposed (5 bit)	cost-0/channel	16236	0	6.532ns	0.0831
	cost-1/channel	18726	0	6.954ns	0.0425
	cost-2/channel	19546	0	7.657ns	0.0324

to ignore the unsafety of certain 5 bit instances. Before doing any synthesis, we can observe in the lower part of Table I the adder cost of the proposed 5 bit instances. The possibility to re-train the network for the cost-0 coefficients, obviously, significantly simplifies the architecture. Training for the cost-1 coefficients still divides the number of used adders by more than 4 w.r.t. the 8 bit QAT baseline. One can also note, that using the cost-2 coefficients does not necessarily lead to doubling the total number of adders, as some channels will still have lower adder cost. Finally, training for the safe cost-1 and cost-2 controllers layer-wise, even if demonstrated possible and pertinent w.r.t. cost, results in a non-parallel architecture, hardware implementation of which we leave to future work.

Multiplications are the resource dominating part of an NNC but not the only part. Hence, we performed synthesis experiments for variants of full NNC implementations. Table III collects the synthesis results for the baseline 8 bit network, implemented in two flavors (generic multipliers with and without DSPs, and directly applying the shift-and-add approach [12] upon coefficients), as well as our multiplierless architectures.

We can observe that the cost-0 architecture divides the number of LUTs by 2.73 compared to the baseline implemented with generic multipliers (without using DSPs) and by 2.59 compared to the state-of-the-art shift-and-add baseline. The reduced complexity also translates to reduced delays: the cost-0 solution is 1.91 and 1.22 times faster than the generic and multiplierless baselines, respectively. As the adder cost predicted, we do not observe a large resource increase going to cost-1 and cost-2 instances. The MSE of our implementation (last column of Table III) was evaluated from the VHDL simulation and is shown to be even less than the FP reference.

VI. Conclusion

We proposed a novel technique for an adder-aware training of small feed-forward DNNs aiming fast parallel inference on FPGAs. Our key idea is that DNNs can be successfully retrained for low-precision coefficients well-suited for a multiplierless implementation with a given target cost. We demonstrated effectiveness of our approach on an NNC implementing automatic cruise control, for which the resource requirements could be divided by 2.59 w.r.t. a state-of-the-art multiplierless hardware implementation obtained with quantization-aware training. Future work includes a thorough analysis of errors due to roundings and incorporation of intermediate truncations, extension to convolutional neural networks and sequential implementations using layer-wise results.

References

[1] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer, "Policy compression for aircraft collision avoidance systems," in *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, 2016, pp. 1–10.

[2] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 2016. [Online]. Available: https://arxiv.org/abs/1604.07316

[3] B. Wu, A. Wan, F. Iandola, P. H. Jin, and K. Keutzer, "Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving," in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017, pp. 446–454.

[4] O. Gustafsson and F. Qureshi, "Addition Aware Quantization for Low Complexity and High Precision Constant Multiplication," *Signal Processing Letters, IEEE*, vol. 17, no. 2, pp. 173 – 176, 2010.

[5] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," 2021. [Online]. Available: https://arxiv.org/abs/2103.13630

[6] J. Faraone, M. Kumm, M. Hardieck, P. Zipf, X. Liu, D. Boland, and P. H. W. Leong, "AddNet: Deep neural networks using FPGA-optimized multipliers," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 28, no. 1, pp. 115 – 128, 00 2020.

[7] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 2722–2730.

[8] H.-D. Tran, X. Yang, D. M. Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson, "Nnv: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems," 2020. [Online]. Available: https://arxiv.org/abs/2004.05519

[9] A. Dempster and M. D. Macleod, "Constant Integer Multiplication Using Minimum Adders," *IEE Proceedings of Circuits, Devices and Systems*, vol. 141, no. 5, pp. 407 – 413, 1994.

[10] Y. Voronenko and M. Püschel, "Multiplierless Multiple Constant Multiplication," *ACM Transactions on Algorithms*, vol. 3, no. 2, pp. 1 – 38, 2007.

[11] O. Gustafsson, "A Difference Based Adder Graph Heuristic for Multiple Constant Multiplication Problems." IEEE International Symposium on Circuits and Systems (ISCAS), 2007, pp. 1097 – 1100.

[12] M. Kumm, P. Zipf, M. Faust, and C.-H. Chang, "Pipelined Adder Graph Optimization for High Speed Multiple Constant Multiplication." IEEE International Symposium on Circuits and Systems, 2012, pp. 49 – 52.

[13] L. Aksoy, E. Günes, and P. Flores, "Search Algorithms for the Multiple Constant Multiplications Problem: Exact and Approximate," *Microprocessors and Microsystems*, vol. 34, no. 5, pp. 151 – 162, 2010.

[14] M. Kumm, "Optimal Constant Multiplication using Integer Linear Programming," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 5, pp. 567 – 571, 2018.

[15] R. Garcia, A. Volkova, and M. Kumm, "Truncated Multiple Constant Multiplication with Minimal Number of Full Adders," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2022.

[16] F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with flopoco," *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 18–27, 2011.

[17] D. M. Lopez, P. Musau, H.-D. Tran, and T. T. Johnson, "Verification of closed-loop systems with neural network controllers," vol. 61, pp. 201–210, 2019.